CROSS ASSEMBLER
FOR THE MCS-48 FAMILY

Produced by

Mumford Micro Systems
Box 400, Summerland, CA 93067
(805) 969-4557

8048 CROSS ASSEMBLER
TABLE OF CONTENTS

Introduction

This 8048 Cross Assembler supports the the Intel MCS-48 family of single chip microcontrollers, which includes a large number of individual components. The MCS-48 family contains five basic groups, each characterized by slightly different instructions sets. This assembler will support all groups and their respective instructions with a pseudo-op to declare which set is desired. In addition, the assembler includes a large number of other pseudo-ops for symbol definition, value storage, conditional assembly, and listing control. It supports standard Intel mnemonics, and includes a full set of arithmetic, logical, and relational operators. Other features include complete expression evaluation, ten significant characters for symbols, and informative error messages.

Source for the assembler may be generated by just about any text editor which will create an ASCII file. Each line must end with a carriage return, except that a line feed may follow. The source file should not contain line numbers or any control characters other than carriage return, line feed, and tab. Source files are assembled directly from disk to an object file on disk. Several switches may be specified when invoking the assembler which affect listing, symbol table generation, and error display.

It is beyond the scope of these instructions to serve as a programming manual for MCS-48 family, but more information on these chips may be obtained in the Intel Component Data Catalog, the Intel Microcontroller User's Manual, and the Intel Microcontroller Applications Handbook, as well as data sheets on the individual components. Literature may be ordered from Intel at the following address:

Intel Corporation
Literature Department SV#-3
3065 Bowers Avenue
Santa Clara, CA 95051

The one component in the MCS-48 family that will be of most interest to the hobbyist or experimenter is probably the 8748. This is a single chip microcomputer with 1024 words of EPROM memory, 64 bytes of RAM memory, 27 I/O lines, an 8 bit timer/counter, and an internal clock oscillator in a single package. It is ideally suited to many control applications that previously would have required many separate components for the processor, program memory, RAM, I/O ports, clock, and address decode logic.

Microcontrollers in this family have found wide useage in many "smart" computer peripherals such as printers, modems, and keyboards, as well as in more mundane consumer appliances like microwave ovens and washing machines. With the availability of a disk-based cross assembler for the 8048 family that will run on the popular TRS-80 Models 1, 3, and 4, it becomes possible for the hobbyist and experimenter to use these readily available and inexpensive components in their own individual dedicated controller applications from phone dialers and print spoolers to solar energy controllers and burglar alarms. Many such projects will require little more than a five volt power supply, an 8748, and a few switches and passive components.

Section 1 - Source Format

1.1 Creation of a source file

The assembler is designed to assemble from a text file which has been previously written with a text editor. There are many word processors available for the TRS-80, and probably any of them will generate an acceptable source file. The source file should contain only ASCII characters, and program lines should not be numbered. Each source line will have four basic fields, though all fields may not be needed on each line. Fields may be separated with a single space, multiple spaces, or tabs.

1.2 The label field

The left-most field is the label field. If a label is used on a line, it must begin with the first character on that line. Labels may contain upper-case letters, lower-case letters, numbers, the underline character, a question mark, or the "at" sign (@). Lower-case letters are considered distinct and different from upper-case letters. The first character of a label may not, however, be a number. Labels may be up to ten characters in length and all characters are significant. If desired, labels may end with a colon which can also serve as a field delimiter for the label field.

1.3 The opcode field

The second field in a source line is the opcode field. A summary of legal opcodes for the various groups in the MCS-48 family is given in Appendix A. Opcodes must be entered in upper-case characters to be recognized by the assembler. Some opcodes will require an operand in the next field, and some opcodes stand alone.

1.4 The operand field

The third field in a source line is the operand field. This field includes any registers referenced by the instruction in the opcode field and any symbols, constants, or expressions required by the opcode. A complete listing of the operands required for each instruction is given in Appendix A. When the operand is an immediate numeric value, arithmetic expressions may be used. Symbols which reference labels or symbols which are equated elsewhere may also be used.

1.5 The comment field

The right-most field in the source line is the comment field. Actually, the comment field may begin anywhere in the source line, but it must always begin with a semicolon, and everything which follows the semicolon will be considered a comment and will be ignored by the assembler.

1.6 Examples

The following lines are typical source lines for the assembler:

```
SYMBOL   OPCODE   OPERAND              COMMENT
START    MOV      A,10                 ;comment
START:   DB       'This is a test'
ABCDEFG: EQU      START+22
         JMP      EXIT  ;comment field can be anywhere to right of operand
PRTDRV:                                ;label with no opcode
         CALL     PRINT                ;
;STORAGE AREA
         MOV      A,(VALUE1+VALUE2) AND MASK
```

In general, the source file will begin with an ORG pseudo-op (see Section 4.12) followed by the body of the program, and terminate with the END statement (see Section 4.4). Once the source file has been written and saved on disk, it is called into the assembler on the "command line" when the assembler is run.

Section 2 - Running The Assembler

The name of the assembler is CASM/CMD. The command line used to run the assembler has the general form:

CASM INPUT [OUTPUT] [-LSTPE] <ENTER>

The first word, CASM, is required. It is the name of the assembler, and when the computer encounters it in the command line the assembler is loaded into memory and begins to run.

2.1 Input file specification

The second word, INPUT, represents the name of your source file. It might really be INPUT, or it might be CONTROL, or TEST/SRC:1, or any other valid filespec you have used for your source file. This is the file which CASM will read and try to assemble into an "object file". You must specify an input file so the assembler will know what to assemble. If no input file name is given, an error message will be generated.

2.2 Output file specification

The third word, OUTPUT, is in brackets because it is optional. It represents the filename you want to give the object file generated by the assembler. The object file is the assembled program created by the assembler which will be programmed into the 8748 (or any other microcontroller you are writing for). You do not have to create an object file every time you run the assembler, however. If you just want to assemble your source to check for errors, or to generate a listing, don't specify an output file name. If no output filename is given in the command line, no output file will be generated. The output filename may be any valid filespec. It might really be OUTPUT, or it might be CONTROL/OBJ, or TEST/ROM:2.

2.3 Switches

The forth expression on the command line is also in brackets because it too is optional. This expression contains the "switches" you might use to give further instructions to the assembler. Switches tell the assembler how you want it to display the assembly. There are four legal switches, and if any one of them is used it must be preceeded by a hyphen (-). If more than one switch is specified, it may be separated from the preceeding one by a space, a comma, a tab, or nothing at all. The legal switches are:

- L - Display a listing of the assembly.
- S - Display the symbol table.
- T - Truncate the object listing to one line per instruction.
- P - Route the listing, error messages, or symbol table to the line printer.
- E - Halt the listing after each error is displayed. (Hit <ENTER> to resume)

The default values are no listing, no symbol table, don't truncate object listing, output to the video screen, and don't halt after errors. Illegal switches will generate an error message and assembly will be aborted.

2.4 Object file format

When an output file has been specified on the command line the assembler will generate a object file on disk. This object file will be a series of bytes which represent the actual object code for the microcontroller. This file is not an Intel hex file or a Radio Shack command file. It is just the bytes needed by the microcontroller to execute as a program. Once this file has been created, it is up to the user to program them into the particular chip they will be run in. Different programmers have different requirements. If your programmer needs a hex file, you will need to generate one from the binary data file created by this assembler. If you built the programmer described in the plans which are available from Mumford Micro Systems, it came with software which is ready to run on the binary file created by this assembler.

2.5 Examples

The last word in the sample command line, <ENTER>, means that you have to hit the ENTER key after typing the previous characters. The following examples will help clarify the various ways of calling up the assembler:

```
CASM FILE1
```

This command will get the input file FILE1, assemble it, and list any errors on the video screen. No source listing will be displayed and no output file will be generated.

```
CASM FILE1 -L
```

This command will get the input file FILE1, assemble it, and list it and any errors on the video screen.

```
CASM FILE1:1 FILE2 -P
```

This command will get the input file FILE1 from drive 1, assemble it, write the assembled code to the output file FILE2, and list any errors on the printer.

```
CASM FILE1 FILE2 -LSTPE
```

This command will get the input file FILE1, assemble it, list it on the printer, print the symbol table on the printer, truncate any multiple byte instructions in the source to a single line in the listing, halt the display after each error message is printed, and write the assembled object code to the output file FILE2.

Section 3 - Assembler Functions

3.1 Symbols

Symbols (or labels) in the assembler can be up to 10 characters in length and may include any upper-case letter, any lower-case letter, any number, an underline character, a question mark, or an "at" sign (@). The first character, however, must not be a number. If your computer can generate lower-case characters, they are considered as different characters than upper-case ones. The label "LOOP", therefore, is distinct and different from "Loop" and "loop", and all three are legal symbols. The following words are all legal symbols and labels:

```
LOOP   testpoint   Loop99   @HERE   UH_WHAT?   A8?@_Z   THIS_TOO??
```

3.2 Numeric constants

The assembler will accept numeric constants in decimal (base 10), hexadecimal (base 16), binary (base 2), or octal (base 8). The default base, or RADIX, is decimal. All other numbers will require a "radix specifier". These specifiers are single letters which must follow any number which is not in the default base (decimal). They are "D" for decimal (yes, it is redundant and unnecessary, but it is allowed), "H" for hexadecimal, "Q" for octal, and "B" for binary. Hexadecimal numbers that begin with a letter must be preceeded with a zero to tell the assembler that they are not symbols. The following numbers are examples of the possible formats:

```
212H        hexadecimal
234Q        octal
11110000B   binary
0FEFEH      hexadecimal
10D         decimal
1234        decimal
```

3.3 Character constants

Character constants are used like numeric constants except that they are specified as an ASCII character in single quotes instead of as a specific number. The assembler will take the ASCII value of the characters in quotes to use in its operations. Double characters may be used to represent 16 bit values. Since the single quote character is used as a delimiter it becomes a little awkward to use the single quote as a character itself. To get around this difficulty, the assembler will interpret two single quotes together as the single ASCII character "single quote" instead of as two delimiters. The following are examples of legal character constants:

```
'A'    =  41H
'AB'   =  4142H
''''   =  27H
```

3.4 Location counter reference

The dollar sign may be used to represent "the current location in memory" for numeric expressions. This is perhaps best explained by example. In the sample source below, the assembler has assembled object code through address 0077H when it encounters an instruction with a dollar sign in the operand field. The next available (or current) address is 0078. The dollar sign will be taken to mean the number 0078 when the expression "$-9" is evaluated, resulting in the value 6DH.

```
ADDRESS  OBJECT CODE        OPCODE   OPERAND
-------------------------------------------
 0075        0A             IN       A,P2
 0076        77             RR       A
 0077        77             RR       A
 0078        EF6D           DJNZ     $-9
```

3.5 Arithmetic, Boolean, and logical operators

CASM has many arithmetic, logical, and relational operators which can be used in numeric expressions. The characters used to represent these operators are defined below:

Arithmetic Operators

```
        +  -->  Addition
        -  -->  Subtraction
        *  -->  Multiplication
        /  -->  Division
 % or MOD  -->  Modulus (remainder of division)
        +  -->  Unary plus (indicates a number is positive)
        -  -->  Unary minus (indicates a number is negative)
```

Boolean Operators

```
 | or  OR  -->  Logical OR
 ^ or XOR  -->  Logical XOR
 & or AND  -->  Logical AND
 ~ or NOT  -->  Unary logical negation
```

Relational Operators

```
        >  -->  Greater than
        <  -->  Less than
        =  -->  Equal to
        >= -->  Greater than or equal to
        <= -->  Less than or equal to
        != -->  Not equal to
```

Order of evaluations precedence

```
        Unary +     Unary -     NOT       (Highest precedence)
        *   /     MOD
        +   -
        <   <=   >   >=   =   !=
        AND
        OR  XOR                           (Lowest precedence)
```

Up to 4 levels of parentheses can be used to change precedence.

Section 4 - Pseudo-Ops

Instructions which are accepted by the assembler but are not part of the instruction set of the microprocessor are called "pseudo-ops". They are instructions for the assembler as opposed to instructions for the microprocessor. Some pseudo-ops require an argument, and some stand alone. The pseudo-ops accepted by this assembler are shown below. In the examples following each pseudo-op, the left-most column represents the actual code generated by the instruction (in hexadecimal), the next column is the pseudo-op, the third column is the argument (if any), and the last column is a comment describing the meaning of each line.

4.1 DB

Define Byte. This instruction places specific numeric values in the object file. It requires an argument which can be a specific number, a string delimited by single quotes, or an arithmetic expression. Commas can be used to define several bytes on one line.

```
1A              DB      1AH             ;SINGLE DEFINITION
41 41 41 41     DB      'AAAA'          ;MULTIPLE CHARACTERS WITHIN STRING
0C 01           DB      12,5-4          ;MULTIPLE ARGUMENTS WITH COMMAS
8D              DB      HIGHBIT+CR      ;ARITHMETIC EXPRESSION
```

4.2 DS

Define Storage. This instruction reserves a number of memory locations for storage. Admittedly, in a ROM-based application like the 8048 family, a define storage pseudo-op is of questionable value. The memory locations which are reserved with this instruction are not left unaltered, however. They are filled with the current value of the "fill character", which is defined under the pseudo-op FILL (see below). This feature allows the programmer to set all unneeded bytes in the object file to the unprogrammed state for the microprocessor. This feature allows a HEX file (required by some EPROM programmers) to be generated from the object file that will define every memory location, yet not program locations that are uneeded.

The argument of the DS pseudo-op is a 16 bit expression, so it may be any number or expression that evaluates to a number in the range 0 to 65535.

```
00 00           DS      5               ;RESERVES 5 BYTES
00 00                                   ;CURRENT FILL CHARACTER IS 00
00
```

4.3 DW

Define Word. This instruction defines two bytes to be placed in the object file. The argument of this operator can be a specific number, an arithmetic expression, or a two character string delimited with single quotes. Commas may be used to define several words on the same line. Note that bytes are not placed in reverse order in the object file, as is the case with assemblers for some microprocessors (like the Z80).

```
01 02           DW      0102H           ;SPECIFIC VALUE
02 04 41 42     DW      0204H,'AB'      ;NOTE THE DOUBLE CHARACTER STRING
02 33           DW      START-END       ;16 BIT ARITHMETIC EXPRESSION
```

4.4 END

The END statement tells the assembler that the end of the source file has been reached. Source files must end with this statement or a NO END FOUND error will be generated. Also, there should be no text following the END statement or the error message DATA FOUND AFTER END will be displayed. The END statement does not require an argument.

```
                END                     ;END OF PROGRAM
```

4.5 ENDIF

The ENDIF pseudo-op is used to terminate a conditional assembly segment. The segment is initiated with the pseudo-op IF, which is described below. All conditional assembly segments which have been initiated with the IF statement must be terminated with the ENDIF statement or the error message IF WITHOUT MATCHING ENDIF STATEMENT will be displayed. When there are multiple nested IF statements, ENDIF will terminate the last IF statement which was encountered. The ENDIF pseudo-op does not require an argument.

```
        FLAG1   EQU     1               ;DEFINE FLAG1 AS TRUE
                IF      FLAG1 = 1       ;INITIATE CONDITIONAL
96 29           JNZ     EXIT            ;CODE GENERATED IF FLAG1 IS TRUE
                ENDIF                   ;END OF CONDITIONAL
```

4.6 ELSE

The ELSE pseudo-op is used to toggle conditional assembly following a IF statement (see IF pseudo-op below). IF statements do not require an ELSE, but one is allowed where it is convenient. The effect of the ELSE statement is to allow the assembler to generate code for the instructions which fall between the ELSE statement and the ENDIF statement when the argument of the original IF condition is false (equal to zero). The ELSE statement must be preceeded by an IF statement or the error message ELSE WITHOUT MATCHING IF STATEMENT will be displayed. The ELSE pseudo-op does not require an argument.

```
        FLAG1   EQU     0               ;DEFINE FLAG1 AS FALSE
                IF      FLAG1           ;INITIATE CONDITIONAL
                JMP     EXIT1           ;CODE NOT GENERATED - FLAG1 IS FALSE
                ELSE                    ;TOGGLE CONDITIONAL
04 3F           JMP     LOOP            ;CODE IS NOW GENERATED
                ENDIF                   ;TERMINATE CONDITIONAL SEGMENT
```

4.7 EQU

Equate symbol. The EQU pseudo-op creates a symbol with a defined value. The EQU statement is preceeded by the symbol you wish to create and followed by an argument which defines the value of the symbol. The value of the argument must be known on the first pass of the assembler or an error message will be displayed. The symbol may be followed by a colon, a space, or a tab.

```
          CR:       EQU     13        ;DEFINE CR AS THE VALUE 13 DECIMAL
          BIGNUM    EQU     9080H     ;DEFINE BIGNUM AS THE VALUE 9080H
          ABCDEFG:  EQU     13H       ;DEFINE ABCDEFG AS THE VALUE 13 HEX
0D                  DB      CR        ;NOW STORE THE VALUE OF CR
90 80               DW      BIGNUM    ;STORE THIS VALUE
00 13               DW      ABCDEFG   ;STORE THIS VALUE
```

4.8 FILL

The FILL pseudo-op defines the character that is used as a filler for areas of memory that are not defined. FILL requires an argument which is the new value of the FILL character. The FILL character is used to pad memory between the last location used during assembly and a subsequent ORG statement, and the memory locations reserved by a DS (define storage) pseudo-op. In addition, when running the CP/M version of this assembler, any characters needed to fill out the last sector of the object file will be the current FILL character.

The default value of the FILL character is zero. The use of a fill character is convenient in that it allows you to choose a character that represents the unprogrammed condition of the ROM you are writing for. This allows you to write code that will skip over previously programmed areas, or leave areas unprogrammed which you may wish to use later. The 8048 family of microprocessors use zero as the unprogrammed condition for all ROM locations.

```
00 00               DS      4                 ;DEFAULT FILL CHARACTER IS ZERO
00 00
                    FILL    0FFH              ;DEFINE 0FFH AS THE FILL CHARACTER
FF FF FF            DS      3                 ;DEFINE STORAGE FILLED WITH NEW VALUE
```

4.9 IF

The IF statement is used to initiate conditional assembly. It requires an argument which is evaluated as true or false. If the argument evaluates as a number which is not zero, it is considered true. If the argument evaluates to zero, it is considered false.

If the argument of an IF statement is true, the assembler will generate code for the instructions which follow, unless an ELSE statement is encountered. If an ELSE is encountered, the assembler will ignore the instructions which follow it and generate no code until an ENDIF statement is encountered.

If the argument of the initial IF statement evaluates as false, the instructions which follow it will be ignored and no code will be generated until an ELSE or ENDIF statement is encountered, at which time the assembler will once again begin to generate code.

IF statements may be nested up to 255 deep. If this limit is exceeded, the error message CAN'T NEST MORE THAN 255 IF STATEMENTS will be displayed. All IF statements must be terminated with an ENDIF statement or the error message IF WITHOUT MATCHING ENDIF will be displayed.

```
        FLAG    EQU     1       ;DEFINE FLAG AS TRUE
                IF      FLAG=1  ;BEGIN CONDITIONAL
14 69           CALL    RDRAM   ;GENERATE CODE - CONDITION IS TRUE
04 0D           JMP     POLALL  ;GENERATE CODE - CONDITION TRUE
                ELSE            ;TOGGLE CONDITIONAL
                CALL    POLCOMP ;NO CODE GENERATED
                JMP     NOTFUL  ;NO CODE GENERATED
                ENDIF           ;TERMINATE CONDITIONAL
```

4.10 LIST

The LIST pseudo-op is used to control the assembly listing. LIST requires an argument, and if the argument evaluates as false (zero), the assembly listing will be supressed. If another LIST statement is encountered with a true (non-zero) argument, the assembly listing will resume. The default condition for LIST is on, or true.

```
        ON      EQU     1       ;DEFINE ON AS TRUE
        OFF     EQU     0       ;DEFINE OFF AS FALSE
                LIST    ON      ;LISTING IS TURNED ON (TRUE ARGUMENT)
14 69           CALL    RDRAM   ;
04 0D           JMP     POLALL  ;
                LIST    OFF     ;LISTING IS TURNED OFF (FALSE ARGUMENT)
```

4.11 MICRO

This statement allows you to define which microprocessor in the MCS-48 family you are writing for. MICRO is followed by an argument which determines the instruction set that will be used by the assembler. The legal arguments, with the microprocessors they support, are shown below:

```
8048  - 8048, 8748, 8748H, 8749, 8049, 87P50, 8050
8041A - 8041A, 8741A, 8042, 8742
8022  - 8022
8021  - 8021
```

The defualt mode for the assembler is 8048, which includes the 8748 and is the component most likely to be used by the hobbyist or experimenter.

```
        MICRO   8048    ;GENERATE CODE FOR AN 8048 OR 8748
        MICRO   8022    ;GENERATE CODE FOR AN 8022
```

4.12 ORG

This statement sets the program orgin. It is followed by an argument which is the address at which you want to begin assembly of the instructions which follow. If there are multiple ORG statements, each one must have an argument that is greater than the location already reached by the assembler or the error message CAN'T ORG BACKWARDS, ORG IGNORED will be displayed and the ORG will be ignored. If there are undefined memory locations between an ORG statement and the memory location reached by previously assembled instructions, the assembler will generate FILL characters (see Section 4.8) for each undetermined memory location.

```
                ORG     0       ;BEGIN ASSEMBLY AT ADDRESS 0
04 06           JMP     ENTRY   ;JUMP TO LOCATION 6
00 00                           ;FILL CHARACTERS GENERATED BETWEEN ORGS
00 00                           ;
                ORG     6       ;BEGIN ASSEMBLY AT ADDRESS 6
23 0C           MOV     A,12    ;PROGRAM CONTINUES HERE
3A              OUTL    P2,A    ;
```

4.13 PAGE

This statement sends a formfeed character to the printer if the listing has been directed to the printer. It requires no argument, it generates no object code, and it sends one formfeed character to the printer every time it is encountered.

```
46 4F   NOTFULL JNT1    POLCOMP
14 69           CALL    RDRAM
14 57           CALL    WRTPRNT
04 0D           JMP     POLLALL
                PAGE            ;FORMFEED SENT TO LINE PRINTER HERE
26 0D   POLCOMP JNT0    POLLALL
08              INS     A,BUS
A8              MOV     R0,A
14 88           CALL    WRTRAM
04 0D           JMP     POLLALL
```

Appendix A - Instruction Set Summary

The list which follows contains the instructions for all members of the MCS-48 family. The list includes the opcode, the operand, a brief description of the instruction, and an indicator as to which microcontrollers each instruction will work with. Each lettered column represents a group of chips, assigned as follows:

A=8048, 8748, 8049, 8749 - B=8041A - C=8041 - D=8022 - E=8021

Opcode	Operand	Description	A	B	C	D	E
ADD	A,#data	Add immediate to A	X	X	X	X	X
ADD	A,Rr	Add register to A (r=0-7)	X	X	X	X	X
ADD	A,@Rr	Add data memory to A (r=0-1)	X	X	X	X	X
ADDC	A,#data	Add immediate with carry	X	X	X	X	X
ADDC	A,Rr	Add register with carry (r=0-7)	X	X	X	X	X
ADDC	A,@Rr	Add data memory with carry (r=0-1)	X	X	X	X	X
ANL	A,#data	And immediate to A	X	X	X	X	X
ANL	A,Rr	And register to A (r=0-7)	X	X	X	X	X
ANL	A,@Rr	And data memory to A (r=0-1)	X	X	X	X	X
ANL	BUS,#data	And immediate to BUS	X				
ANL	Pp,#data	And immediate to port (p=1-2)	X	X	X		
ANLD	Pp,A	And A to expander port (p=4-7)	X	X	X	X	X
CALL	addr	Call subroutine	X	X	X	X	X
CLR	A	Clear A	X	X	X	X	X
CLR	C	Clear carry flag	X	X	X	X	X
CLR	F0	Clear flag 0	X	X	X		
CLR	F1	Clear flag 1	X	X	X		
CPL	A	Complement A	X	X	X	X	X
CPL	C	Complement carry flag	X	X	X	X	X
CPL	F0	Complement flag 0	X	X	X		
CPL	F1	Complement flag 1	X	X	X		
DA	A	Decimal adjust A	X	X	X	X	X
DEC	A	Decrement A	X	X	X	X	X
DEC	Rr	Decrement register (r=0-7)	X	X	X		
DIS	I	Disable external interrupt	X	X	X	X	
DIS	TCNTI	Disable timer/counter interrupt	X	X	X	X	
DJNZ	Rr,addr	Decrement register and jump (r=0-7)	X	X	X	X	X
EN	DMA	Enable DMA handshaking lines		X			
EN	FLAGS	Enable master interrupts		X			
EN	I	Enable external interrupt	X	X	X	X	
EN	TCNTI	Enable timer/counter interrupt	X	X	X	X	
ENTO	CLK	Enable clock output on line T0	X				
IN	A,DBB	Input DBB to A, clear IBF		X	X		
IN	A,P0	Input port 0 to A				X	X
IN	A,Pp	Input port to A (p=1-2)	X	X	X	X	X
INC	A	Increment A	X	X	X	X	X
INC	Rr	Increment register (r=0-7)	X	X	X	X	X
INC	@Rr	Increment data memory (r=0-1)	X	X	X	X	X
INS	A,BUS	Input BUS to A	X				

Opcode	Operand	Description	A	B	C	D	E
JBb	addr	Jump on accumulator bit (b=0-7)	X	X	X		
JC	addr	Jump on carry flag = 1	X	X	X	X	X
JF0	addr	Jump on F0 flag = 1	X	X	X		
JF1	addr	Jump on F1 flag = 1	X	X	X		
JMP	addr	Jump unconditional	X	X	X	X	X
JMPP	@A	Jump indirect	X	X	X	X	X
JNC	addr	Jump on carry flag = 0	X	X	X	X	X
JNI	addr	Jump on external interrupt = 0	X				
JNIBF	addr	Jump on IBF flag = 0		X	X		
JNT0	addr	Jump on T0 = 0	X	X	X	X	
JNT1	addr	Jump on T1 = 0	X	X	X	X	
JNZ	addr	Jump on A not Zero	X	X	X	X	X
JOBF	addr	Jump on OBF flag = 1		X	X		
JTF	addr	Jump on timer flag = 1	X	X	X	X	X
JT0	addr	Jump on T0 = 1	X	X	X	X	
JT1	addr	Jump on T1 = 1	X	X	X	X	
JZ	addr	Jump on A Zero	X	X	X	X	X
MOV	A,#data	Move immediate to A	X	X	X	X	X
MOV	A,PSW	Move PSW to A	X	X	X		
MOV	A,Rr	Move register to A (r=0-7)	X	X	X	X	X
MOV	A,@Rr	Move data memory to A (r=0-1)	X	X	X	X	X
MOV	A,T	Read timer/counter	X	X	X	X	X
MOV	PSW,A	Move A to PSW	X	X	X		
MOV	Rr,A	Move A to register (r=0-7)	X	X	X	X	X
MOV	Rr,#data	Move immediate to register (r=0-7)	X	X	X	X	X
MOV	@Rr,A	Move A to data memory (r=0-1)	X	X	X	X	X
MOV	@Rr,#data	Move immediate to data memory	X	X	X	X	X
MOV	STS,A	A4-A7 to bits 4-7 of status		X			
MOV	T,A	Load timer/counter	X	X	X	X	X
MOVD	A,Pp	Input expander port to A (p=4-7)	X	X	X	X	X
MOVD	Pp,A	Output A to expander port (p=4-7)	X	X	X	X	X
MOVP	A,@A	Move to A from current page	X	X	X	X	X
MOVP3	A,@A	Move to A from page 3	X	X	X		
MOVX	A,@Rr	Move external data memory to A (r=0-1)	X				
MOVX	Rr,A	Move A to external data memory (r=0-1)	X				
NOP		No operation	X	X	X	X	X
ORL	A,#data	Or immediate to A	X	X	X	X	X
ORL	A,Rr	Or register to A (r=0-7)	X	X	X	X	X
ORL	A,@Rr	Or data memory to A (r=0-1)	X	X	X	X	X
ORL	BUS,#data	Or immediate to BUS	X				
ORL	Pp,#data	Or immediate to port (p=1-2)	X	X	X		
ORLD	Pp,A	Or A to expander port (p=4-7)	X	X	X	X	X
OUT	DBB,A	Output A to DBB, set OBF		X	X		
OUTL	BUS,A	Output A to BUS	X				
OUTL	P0,A	Output A to port 0				X	X
OUTL	Pp,A	Output A to port (p=1-2)	X	X	X	X	X

Opcode	Operand	Description	A	B	C	D	E
RAD		Move conversion result to A				X	
RET		Return from CALL	X	X	X	X	X
RETI		Return from interrupt				X	
RETR		Return and restore status	X	X	X		
RL	A	Rotate A left	X	X	X	X	X
RLC	A	Rotate A left through carry flag	X	X	X	X	X
RR	A	Rotate A right	X	X	X	X	X
RRC	A	Rotate A right through carry flag	X	X	X	X	X
SEL	AN0	Select analog input 0				X	
SEL	AN1	Select analog input 1				X	
SEL	MB0	Select memory bank 0	X				
SEL	MB1	Select memory bank 1	X				
SEL	RB0	Select register bank 0	X	X	X		
SEL	RB1	Select register bank 1	X	X	X		
STOP	TCNT	Stop timer/counter	X	X	X	X	X
STRT	CNT	Start counter	X	X	X	X	X
STRT	T	Start timer	X	X	X	X	X
SWAP	A	Swap nibbles of A	X	X	X	X	X
XCH	A,Rr	Exchange A and register (r=0-7)	X	X	X	X	X
XCH	A,@Rr	Exchange A and data memory (r=0-1)	X	X	X	X	X
XCHD	A,@Rr	Exchange nibble of A and data memory	X	X	X	X	X
XRL	A,#data	Exclusive or immediate to A	X	X	X	X	X
XRL	A,Rr	Exclusive or register to A (r=0-7)	X	X	X	X	X
XRL	A,@Rr	Exclusive or data memory to A (r=0-1)	X	X	X	X	X

Appendix B - Instruction Mnemonics and Opcodes

The following listing is an assembly of a source file which was written to test the assembler by using every instruction and all addressing modes. This printout is also an example of the listing format of the assembler. The left hand column shows the address, followed by the opcode(s). The right hand two columns are the instruction mnemonics.

```
                    ;instructions for the 8048
                            MICRO   8048
                    ;
0000  68                    ADD     A,R0
0001  69                    ADD     A,R1
0002  6A                    ADD     A,R2
0003  6B                    ADD     A,R3
0004  6C                    ADD     A,R4
0005  6D                    ADD     A,R5
0006  6E                    ADD     A,R6
0007  6F                    ADD     A,R7
0008  60                    ADD     A,@R0
0009  61                    ADD     A,@R1
000A  0355                  ADD     A,#55H
000C  78                    ADDC    A,R0
000D  79                    ADDC    A,R1
000E  7A                    ADDC    A,R2
000F  7B                    ADDC    A,R3
0010  7C                    ADDC    A,R4
0011  7D                    ADDC    A,R5
0012  7E                    ADDC    A,R6
0013  7F                    ADDC    A,R7
0014  70                    ADDC    A,@R0
0015  71                    ADDC    A,@R1
0016  1355                  ADDC    A,#55H
0018  58                    ANL     A,R0
0019  59                    ANL     A,R1
001A  5A                    ANL     A,R2
001B  5B                    ANL     A,R3
001C  5C                    ANL     A,R4
001D  5D                    ANL     A,R5
001E  5E                    ANL     A,R6
001F  5F                    ANL     A,R7
0020  50                    ANL     A,@R0
0021  51                    ANL     A,@R1
0022  5355                  ANL     A,#55H
0024  9855                  ANL     BUS,#55H
0026  9955                  ANL     P1,#55H
0028  9A55                  ANL     P2,#55H
002A  9C                    ANLD    P4,A
002B  9D                    ANLD    P5,A
002C  9E                    ANLD    P6,A
002D  9F                    ANLD    P7,A
002E  1455                  CALL    55H
0030  B455                  CALL    555H
```

```
0032  27          CLR     A
0033  97          CLR     C
0034  A5          CLR     F1
0035  85          CLR     F0
0036  37          CPL     A
0037  A7          CPL     C
0038  95          CPL     F0
0039  B5          CPL     F1
003A  57          DA      A
003B  07          DEC     A
003C  C8          DEC     R0
003D  C9          DEC     R1
003E  CA          DEC     R2
003F  CB          DEC     R3
0040  CC          DEC     R4
0041  CD          DEC     R5
0042  CE          DEC     R6
0043  CF          DEC     R7
0044  15          DIS     I
0045  35          DIS     TCNTI
0046  E855        DJNZ    R0,55H
0048  E955        DJNZ    R1,55H
004A  EA55        DJNZ    R2,55H
004C  EB55        DJNZ    R3,55H
004E  EC55        DJNZ    R4,55H
0050  ED55        DJNZ    R5,55H
0052  EE55        DJNZ    R6,55H
0054  EF55        DJNZ    R7,55H
0056  05          EN      I
0057  25          EN      TCNTI
0058  75          ENT0    CLK
0059  09          IN      A,P1
005A  0A          IN      A,P2
005B  17          INC     A
005C  18          INC     R0
005D  19          INC     R1
005E  1A          INC     R2
005F  1B          INC     R3
0060  1C          INC     R4
0061  1D          INC     R5
0062  1E          INC     R6
0063  1F          INC     R7
0064  10          INC     @R0
0065  11          INC     @R1
0066  08          INS     A,BUS
0067  1255        JB0     55H
0069  3255        JB1     55H
006B  5255        JB2     55H
006D  7255        JB3     55H
006F  9255        JB4     55H
0071  B255        JB5     55H
0073  D255        JB6     55H
0075  F255        JB7     55H
0077  F655        JC      55H
```

```
0079  B655        JF0     55H
007B  7655        JF1     55H
007D  0455        JMP     55H
007F  A455        JMP     555H
0081  B3          JMPP    @A
0082  E655        JNC     55H
0084  8655        JNI     55H
0086  2655        JNT0    55H
0088  4655        JNT1    55H
008A  9655        JNZ     55H
008C  1655        JTF     55H
008E  3655        JT0     55H
0090  5655        JT1     55H
0092  C655        JZ      55H
0094  2355        MOV     A,#55H
0096  C7          MOV     A,PSW
0097  F8          MOV     A,R0
0098  F9          MOV     A,R1
0099  FA          MOV     A,R2
009A  FB          MOV     A,R3
009B  FC          MOV     A,R4
009C  FD          MOV     A,R5
009D  FE          MOV     A,R6
009E  FF          MOV     A,R7
009F  F0          MOV     A,@R0
00A0  F1          MOV     A,@R1
00A1  42          MOV     A,T
00A2  D7          MOV     PSW,A
00A3  A8          MOV     R0,A
00A4  A9          MOV     R1,A
00A5  AA          MOV     R2,A
00A6  AB          MOV     R3,A
00A7  AC          MOV     R4,A
00A8  AD          MOV     R5,A
00A9  AE          MOV     R6,A
00AA  AF          MOV     R7,A
00AB  B855        MOV     R0,#55H
00AD  B955        MOV     R1,#55H
00AF  BA55        MOV     R2,#55H
00B1  BB55        MOV     R3,#55H
00B3  BC55        MOV     R4,#55H
00B5  BD55        MOV     R5,#55H
00B7  BE55        MOV     R6,#55H
00B9  BF55        MOV     R7,#55H
00BB  A0          MOV     @R0,A
00BC  A1          MOV     @R1,A
00BD  B055        MOV     @R0,#55H
00BF  B155        MOV     @R1,#55H
00C1  62          MOV     T,A
00C2  0C          MOVD    A,P4
00C3  0D          MOVD    A,P5
00C4  0E          MOVD    A,P6
00C5  0F          MOVD    A,P7
00C6  3C          MOVD    P4,A
```

```
00C7  3D          MOVD    P5,A
00C8  3E          MOVD    P6,A
00C9  3F          MOVD    P7,A
00CA  A3          MOVP    A,@A
00CB  E3          MOVP3   A,@A
00CC  80          MOVX    A,@R0
00CD  81          MOVX    A,@R1
00CE  90          MOVX    @R0,A
00CF  91          MOVX    @R1,A
00D0  00          NOP
00D1  48          ORL     A,R0
00D2  49          ORL     A,R1
00D3  4A          ORL     A,R2
00D4  4B          ORL     A,R3
00D5  4C          ORL     A,R4
00D6  4D          ORL     A,R5
00D7  4E          ORL     A,R6
00D8  4F          ORL     A,R7
00D9  40          ORL     A,@R0
00DA  41          ORL     A,@R1
00DB  4355        ORL     A,#55H
00DD  8855        ORL     BUS,#55H
00DF  8955        ORL     P1,#55H
00E1  8A55        ORL     P2,#55H
00E3  8C          ORLD    P4,A
00E4  8D          ORLD    P5,A
00E5  8E          ORLD    P6,A
00E6  8F          ORLD    P7,A
00E7  02          OUTL    BUS,A
00E8  39          OUTL    P1,A
00E9  3A          OUTL    P2,A
00EA  83          RET
00EB  93          RETR
00EC  E7          RL      A
00ED  F7          RLC     A
00EE  77          RR      A
00EF  67          RRC     A
00F0  E5          SEL     MB0
00F1  F5          SEL     MB1
00F2  C5          SEL     RB0
00F3  D5          SEL     RB1
00F4  65          STOP    TCNT
00F5  45          STRT    CNT
00F6  55          STRT    T
00F7  47          SWAP    A
00F8  28          XCH     A,R0
00F9  29          XCH     A,R1
00FA  2A          XCH     A,R2
00FB  2B          XCH     A,R3
00FC  2C          XCH     A,R4
00FD  2D          XCH     A,R5
00FE  2E          XCH     A,R6
00FF  2F          XCH     A,R7
0100  20          XCH     A,@R0
```

```
0101  21              XCH     A,@R1
0102  30              XCHD    A,@R0
0103  31              XCHD    A,@R1
0104  D8              XRL     A,R0
0105  D9              XRL     A,R1
0106  DA              XRL     A,R2
0107  DB              XRL     A,R3
0108  DC              XRL     A,R4
0109  DD              XRL     A,R5
010A  DE              XRL     A,R6
010B  DF              XRL     A,R7
010C  D0              XRL     A,@R0
010D  D1              XRL     A,@R1
010E  D355            XRL     A,#55H
                ;
                ;instructions for the 8041A
                      MICRO   8041A
0110  E5              EN      DMA
0111  F5              EN      FLAGS
0112  22              IN      A,DBB
0113  D655            JNIBF   155H
0115  8655            JOBF    155H
0117  90              MOV     STS,A
0118  02              OUT     DBB,A
                ;
                ;instructions for the 8022
                      MICRO   8022
0119  80              RAD
011A  93              RETI
011B  85              SEL     AN0
011C  95              SEL     AN1
                ;
                ;instructions for the 8021
                      MICRO   8021
011D  08              IN      A,P0
011E  90              OUTL    P0,A
                ;--------------------------------
                ;             Expressions
                ;--------------------------------
C33C            STAN  EQU     0C33CH
6699            FRED  EQU     6699H
0001            L1    EQU     1
0001            L2    EQU     1B
0001            L3    EQU     1D
0001            L4    EQU     1Q
0001            L5    EQU     1H
007B            L6    EQU     123D
0096            L7    EQU     10010110B
0141            L8    EQU     321D
0305            L9    EQU     773
1234            L10   EQU     1234H
011F            L11   EQU     $
0053            L12   EQU     'S'
5352            L13   EQU     'SR'
```

```
5327            L14     EQU     'S'''
2753            L15     EQU     '''S'
C33C            L16     EQU     STAN
FFFE            L17     EQU     -2
FFFE            L18     EQU     - 2
FEE1            L19     EQU     -$
3CC4            L20     EQU     -STAN
FFAD            L21     EQU     -'S'
ACAE            L22     EQU     -'SR'
FF34            L23     EQU     -11001100B
FFFD            L24     EQU     NOT 2
0002            L25     EQU     +2
E7BD            L26     EQU     STAN OR FRED
E7BD            L27     EQU     STAN | FRED
A5A5            L28     EQU     STAN XOR FRED
4218            L29     EQU     STAN AND FRED
29D5            L30     EQU     STAN + FRED
5CA3            L31     EQU     STAN - FRED
8678            L32     EQU     STAN * 2
619E            L33     EQU     STAN / 2
003C            L34     EQU     STAN MOD 100H
FFFF            L35     EQU     STAN = STAN
0000            L36     EQU     STAN = FRED
FFFF            L37     EQU     STAN > FRED
0000            L38     EQU     FRED > STAN
0000            L39     EQU     STAN > STAN
0000            L40     EQU     STAN < FRED
FFFF            L41     EQU     FRED < STAN
0000            L42     EQU     STAN < STAN
FFFF            L43     EQU     STAN >= FRED
0000            L44     EQU     FRED >= STAN
FFFF            L45     EQU     STAN >= STAN
0000            L46     EQU     STAN <= FRED
FFFF            L47     EQU     FRED <= STAN
FFFF            L48     EQU     STAN <= STAN
FFFF            L49     EQU     STAN != FRED
0000            L50     EQU     STAN != STAN
C33C            L51     EQU     (STAN)
C33C            L52     EQU     ( STAN )
C33C            L53     EQU     ((((STAN))))
0012            L54     EQU     3 * (4 + 2)
000E            L55     EQU     (3 * 4) + 2
000E            L56     EQU     3 * 4 + 2
0000            L58     EQU     0 / STAN
0001            L59     EQU     STAN / STAN
0000            L60     EQU     1 / STAN
```

Appendix C - Error Messages

Branch out of page boundary - An attempt has been made to do a conditional jump into another page.

Byte value truncated - An attempt has been made to use a 16 bit value where an 8 bit value is required.

Can't ORG backwards, ORG ignored - An ORG statement has been used which specifies an address lower than that which has already been reached by the assembler (see Section 4.12).

Can't nest more than 255 IF statements - An attempt has been made to use more than 255 unterminated IF statements simultaneously (see Section 4.9).

Can't open INPUT file - The source file does not exist or a disk I/O error has occurred.

Can't open OUTPUT file - The disk is write protected or a disk I/O error has occurred.

Data found after END - The END pseudo-op is not the last statement in the source file (see Section 4.4).

Disk CLOSE error - A disk I/O error has occurred while closing the output file.

Disk WRITE error - A disk I/O error has occurred while writing the output file.

Division by zero - An expression has been evaluated to a condition where a number is being divided by zero.

Doubly defined label - An attempt has been made to define the same label more than once.

ELSE without matching IF statement - An ELSE statement has been encountered without a previously defined IF statement (see Sections 4.9 and 4.6).

ENDIF without matching IF statement - An ENDIF statement has been encountered without a previously defined IF statement (see Sections 4.9 and 4.5).

IF without matching ENDIF statement - An IF statement has not been terminated by an ENDIF before the end of the source file was encountered (see Sections 4.9 and 4.5).

Illegal label - An attempt has been made to use a label which does not meet the conditions defined under Section 3.1.

Illegal number - An attempt has been made to use a number which is either too large or contains illegal characters (see section 3.2).

Input file not specified or illegal - The command line does not contain a properly specified source file name (see Section 2.1).

Invalid opcode - The instruction is not valid for the processor defined by the MICRO pseudo-op. See Appendix A for a list of valid opcodes.

Invalid operand - The operand is either missing, illegal, or specified incorrectly for the preceeding instruction (see Section 1.4 and Appendix A).

Invalid short string - A syntax error has occurred during the specification of a single or double byte character constant (see Section 3.3).

Invalid string - An uneven number of delimiters has been used when defining a string (see Section 3.3).

Invalid switch - A syntax error has occurred in the switch portion of the command line (see Section 2.3).

No END found - No END pseudo-op was found at the end of the source file (see Section 4.4).

Out of symbol table memory - There is insufficient memory to perform the assembly.

Parentheses nested too deep - An attempt has been made to have more than four levels of parentheses open simultaneously.

Parentheses uneven - An expression has been used which contains an open parenthesis without a matching closed parenthesis.

Phase error, value changed on pass 2 - The value of a symbol determined on the first pass of the assembler is not the same as the value arrived at on the second pass.

Ran out of memory during symbol table sort - There is insufficient memory to perform the assembly.

Stack overflow - An expression has been used which cannot be evaluated.

Symbol not defined on pass 1 - A symbol has been used which references a label which was not defined on the first pass of the assembler.

Undefined symbol - An instruction references a label which has not been defined elsewhere in the source.

Appendix D - Sample I/O and Math Routines

```
;       ************************************************
;       *                                              *
;       *          Serial I/O and Math routines        *
;       *                                              *
;       ************************************************
;
RS232   EQU     00001000B       ;serial output bit on port 2
;
; The delay constant for serial data rate is computed with:
;       t = (400,000 / BAUD - 24) / 6
;
Bd9600  EQU     ((40000 / (9600 / 100) + 5) / 10 - 24) / 6
Bd1200  EQU     ((40000 / (1200 / 100) + 5) / 10 - 24) / 6
Bd300   EQU     ((40000 / (300  / 100) + 5) / 10 - 24) / 6
;
;
;call to read a byte from serial port, data input on T1
;       Enter:  R0' = delay constant
;       Exit:   A = byte
;               R7,R7' = undefined
SerIn   MOV     R7,#8           ;read in 8 bits
SerIn1  JNT1    SerIn1          ;wait till line goes high
SerIn2  JT1     SerIn2          ;wait till line goes low
;
        CALL    HlfDly          ;wait for half a bit
        JT1     SerIn           ;branch if start bit missing
;
SerIn3  CALL    BitDly2         ;delay for one bit
;
        JNT1    SerIn4          ;branch if data bit low
        CLR     C               ;clear the carry
        CPL     C
        RRC     A               ;rotate the data into A
        DJNZ    R7,SerIn3       ;dcr bit count, test for done
        RET
;
SerIn4  CLR     C               ;set the carry bit
        NOP                     ;keep the timing right
        RRC     A               ;rotate the data into A
        DJNZ    R7,SerIn3       ;dcr bit count, test for done
        RET
```

```
;call SerOut to set a byte out the serial port
;       Enter:  A = byte to be sent
;               R0' = delay constant
;       Exit:   A = byte sent out
;               R7,R7' = undefined
SerOut  MOV     R7,#8           ;transmit 8 bits
;
        ANL     P2,#~RS232      ;send the start bit
        CALL    BitDly1         ;delay for 1 bit time
;
SerOut1 RR      A               ;rotate to get next bit
        JB7     SerOut2         ;branch if bit is a one
;
        ANL     P2,#~RS232      ;set the bit low
        JMP     SerOut3
;
SerOut2 ORL     P2,#RS232       ;set the bit high
        JMP     SerOut3         ;keep the timing straight
;
SerOut3 CALL    BitDly3         ;delay one bit time
        DJNZ    R7,SerOut1      ;go do next bit if not done
;
        ORL     P2,#RS232       ;send the stop bit
        CALL    BitDly1         ;delay one bit time
        RET
;
;
;call to delay one serial bit time
;       Enter:  R0' = delay constant
;       Exit:   R7' = undefined
BitDly1 NOP                     ;delay for 8 cycles first
        NOP
BitDly2 NOP                     ;delay for 6 cycles first
        NOP
BitDly3 NOP                     ;delay for 4 cycles first
        NOP
        NOP
        NOP
        SEL     RB1             ;select second register bank
        MOV     R7,A            ;save A in R7'
        MOV     A,R0            ;get the time delay constant
BitDly4 NOP                     ;delay loop time = 6 cycles
        NOP                     ;nops delay for 3
        NOP
        DEC     A               ;dec the loop counter
        JNZ     BitDly4         ;loop till done
;
        MOV     A,R7            ;restore A
        SEL     RB0             ;select first register bank
        RET
```

```
;call to delay a half serial bit time
;       Enter:  R0' = delay constant
;       Exit:   R7' = undefined
HlfDly  SEL     RB1             ;select second register bank
        MOV     R7,A            ;save A in R7'
        MOV     A,R0            ;get the time delay constant
;
HlfDly1 DEC     A               ;dec the loop counter
        JNZ     HlfDly1         ;loop till done
;
        MOV     A,R7            ;restore A
        SEL     RB0             ;select first register bank
        RET
;
;---------------------------------------------------------
;       Math Utilities
;---------------------------------------------------------
;
;call to do a double subtraction
;       Enter:  R1,R2 = Minuend, R1 = MSB
;               R3,R4 = Subtrahend, R3 = MSB
;       Exit:   R3,R4 = difference, R3 = MSB
;               R1,R2 = Minuend, R1 = MSB
Subtrct MOV     A,R2            ;subtract R4 from R2
        CPL     A
        ADD     A,R4
        CPL     A
        MOV     R4,A            ;put difference in R4
        MOV     A,R1            ;subtract R3 from R1
        CPL     A
        ADDC    A,R3            ;include carry from above
        CPL     A
        MOV     R3,A            ;put difference in R3
        RET
```

```
;call to do a 8 x 16 multiply, product 24 bits
;       Enter:  R1 = multiplicand #1
;               R2,R3 = multiplicand #2, R2 = MSB
;       Exit:   R4,R5,R6 = product, R4 = MSB
;               R2,R3 = multiplicand #2, R2 = MSB
;               R1,R7 = undefined
Multply MOV     R4,#0           ;start with a total of 0
        MOV     R5,#0
        MOV     R6,#0           ;LSB
;
        MOV     R7,#8           ;multiply all 8 bits
Mult1   MOV     A,R6            ;double the product
        CLR     C
        RLC     A               ;rotate the low byte
        MOV     R6,A
        MOV     A,R5            ;rotate the middle byte
        RLC     A
        MOV     R5,A
        MOV     A,R4            ;rotate the high byte
        RLC     A
        MOV     R4,A
;
        MOV     A,R1            ;shift multiplicand
        CLR     C
        RLC     A
        MOV     R1,A
        JNC     Mult2           ;if high bit 0 don't add on
;
        MOV     A,R6            ;add multiplicand to product
        ADD     A,R3            ;add low bytes
        MOV     R6,A
        MOV     A,R5            ;add middle bytes
        ADDC    A,R2
        MOV     R5,A
        MOV     A,R4            ;add on carry to high byte
        ADDC    A,#0
        MOV     R4,A
;
Mult2   DJNZ    R7,Mult1        ;go multiply next bit
        RET                     ;all done
```

```
;call to divide a 24 bit number by a 8 bit one
;       Enter:  R4, R5, R6 = dividend, R4 = MSB
;               R3 = divisor
;       Exit:   R0, R1, R2 = quotient, R0 = MSB
;               R4, R5, R6 = remainder * 2, R4 = MSB
;               R3,R7 = undefined
Divide  MOV     R0,#0           ;start with quotient = 0
        MOV     R1,#0
        MOV     R2,#0
;
        MOV     A,R3            ;negate divisor
        CPL     A
        INC     A
        MOV     R3,A            ;R3 = -divisor
;
        MOV     R7,#16+1        ;divide 24 bit number
Div1    MOV     A,R2            ;shift quotient left once
        CLR     C               ;quotient = quotient * 2
        RLC     A               ;shift the low byte
        MOV     R2,A
        MOV     A,R1            ;shift the middle byte
        RLC     A
        MOV     R1,A
        MOV     A,R0            ;shift the high byte
        RLC     A
        MOV     R0,A
;
        MOV     A,R3            ;subtract divisor from dividend
        ADD     A,R4            ;dividend bits 23 - 16
        JNC     Div2            ;jmp dividend[23-16] < divisor
;
        MOV     R4,A            ;Div[23-16]=Div[23-16]-divisor
        INC     R2              ;set bit in quotient
;
Div2    MOV     A,R6            ;shift dividend left once
        CLR     C               ;dividend = dividend * 2
        RLC     A               ;shift the low byte
        MOV     R6,A
        MOV     A,R5            ;shift the middle byte
        RLC     A
        MOV     R5,A
        MOV     A,R4            ;shift the high byte
        RLC     A
        MOV     R4,A
;
        DJNZ    R7,Div1         ;go divide next bit
        RET
```

```
;call to negate a 16 bit number
;       Enter:  R2,R3 = binary # to negate, R2 = MSB
;       Exit:   R2,R3 = negated number
Negate  MOV     A,R3            ;get the low byte
        CPL     A               ;invert it
        ADD     A,#1            ;increment it
        MOV     R3,A            ;R3 = low byte
        MOV     A,R2            ;get high byte
        CPL     A               ;invert it
        ADDC    A,#0            ;add carry from before
        MOV     R2,A            ;R2 = high byte
        RET
;
;
;
;call to convert a 16 bit number to a 4 digit BCD number
;       Enter:  R3,R4 = binary # to be changed to BCD, R3 = MSB
;       Exit:   R3,R4 = number in BCD, R3 = MSB
;               R5,R6,R7 = undefined
CnvBcd  MOV     R5,#0           ;zero the 4 digit BCD array
        MOV     R6,#0
;
        MOV     R7,#16          ;convert a 16 bit number
CnvBcd1 MOV     A,R4            ;shift the binary number left
        CLR     C               ;first the low byte
        RLC     A
        MOV     R4,A
        MOV     A,R3            ;then the high byte
        RLC     A
        MOV     R3,A
;
        MOV     A,R6            ;double low BCD # + carry
        ADDC    A,R6
        DA      A               ;decimal fix it
        MOV     R6,A            ;save the new BCD value
        MOV     A,R5            ;double high BCD # + carry
        ADDC    A,R5
        DA      A               ;decimal fix it
        MOV     R5,A            ;save the new BCD value
        DJNZ    R7,CnvBcd1      ;go stick on next bit of #
;
        MOV     A,R5            ;put result in R3 & R4
        MOV     R3,A
        MOV     A,R6
        MOV     R4,A
        RET
```